



Getting Started

Nobl9 TechDoc: Getting Started

Back to [Nobl9 Documentation](#)

Welcome, and thank you for choosing Nobl9. This Getting Started Guide shows you how to start using the Nobl9 platform. Learn to access your account, connect to metrics, and work through your first service level objective (SLO).

Prerequisites

The following software, tools, or actions are needed to ensure a great onboarding experience:

1. Use a Mac, Linux, or Windows machine to run `sloctl`. The Nobl9 command-line utility makes it easier to create and update many SLOs at once.
2. Select a Kubernetes cluster or any [Docker environment](#) (or use a Docker environment on your local machine to start) to run the Nobl9 agent, which collects service level indicator metrics from your existing metrics system (such as Datadog, New Relic, or Prometheus).

💡 *Soon you will be able to run Nobl9 without an agent as well for certain data stores.*

3. Verify that you received an email from Nobl9 to set up a user account.

Setting Up a User Account

As a Nobl9 customer, you will receive an invitation email with an activation link.

💡 *If you were invited to Nobl9 and did not receive an invitation email, please contact support@nobl9.com.*

1. Locate the Nobl9 user invitation sent to your email.
2. Click the link to accept the invitation and follow the instructions to set up your user account. After setting up your account, a confirmation page appears which asks you to return to the login screen.
3. Return to the login screen by visiting <https://app.nobl9.com> in your browser.

Logging into Nobl9 User Interface

You will need to log into the Nobl9 web user interface (UI) using the credentials created during the account setup.

1. Go to <https://app.nobl9.com>
2. Enter your email address and password created during the account setup, or click Login with Google if you have a single sign-on (SSO) account.

Setting Up Sloctl

Use the following steps to install and configure Nobl9 command-line interface, sloctl.

Binary Installation



When installing files in protected folders, the operating system occasionally requires copy or file permissions. When this happens, give the installed files executable permissions (Linux and Mac) or confirm the file copy operation (Windows).

1. For **Mac**, we recommend using brew:

- [Install Homebrew](#)
- Use Terminal to complete the installation:

```
brew tap nobl9/sloctl  
brew install sloctl
```

2. For **Windows** and **Linux**, we recommend manual installation from an archive:

- Download the appropriate binary executable zip file from <https://github.com/nobl9/sloctl/releases>.
- Extract the binary. Copy the executable file into the following platform-specific location:
 - **Linux:** /usr/local/bin
|  For further details, see [Linux instructions](#).
 - **Windows:** C:\Windows\System32
|  For further details and an example of installing into a separate program folder, see [Windows instructions](#).


Configuration

1. Create a **Client ID** and **Client Secret** pair for use in sloctl.

- Navigate to **Settings** → **Access Keys** in the web UI.
 - Click **Create Access Key**.
2. Follow UI instructions to configure slocctl to use the provided credentials. Use one of the available set-up flows:
 - Click the **Download credentials file** in the web UI and put downloaded file in `~/.config/nobl9/config.toml` (Linux and macOS) or `%UserProfile%\config\nobl9\config.toml` (Windows).

or

- Run `slocctl add-context`, name the context, and paste the **Client ID** and **Client Secret** from the web UI when prompted.
3. Test the configuration by entering `slocctl get slos` into the command line.

 *If there are no SLOs created in your account or in the selected project, you might see this message: **No resources found in default project.** The message means the configuration is correct. The command line will return a **401 error** if the configuration does not work.*

Defining a Data Source and Running a Nobl9 Agent

Running data collection through an agent means that special inbound access to your network is not needed and Nobl9 doesn't have to store credentials to your other metric systems.

Use the following steps to define a data source and run a Nobl9 Agent.

1. Go to **Integrations** icon in web UI.
2. Select the **Sources** tab to define a data source.
3. Follow the on-screen instructions to run the agent.

Recommendations:

1. Samples are provided for a Kubernetes Deployment and a simple Docker run command
2. Run the agent(s) in production clusters or in a location that can access production metrics
3. Consider running the agent in your local Docker environment at first for ease of troubleshooting

Agent	Access
New Relic	The agent just needs internet access.
Datadog	The agent just needs internet access.

Prometheus	The agent needs direct access to the Prometheus instance that it queries.
AppDynamics	The agent needs direct access to the AppDynamics instance that it queries.
Splunk	The agent needs direct access to the Splunk instance that it queries.
Lightstep	The agent needs direct access to the Lightstep instance that it queries.
Splunk Observability	The agent needs direct access to the Splunk Observability instance that it queries.

 **Note:** Direct integration available soon.

Add Services and SLOs

1. Add service

Nobl9 uses services to represent distinct boundaries in your application. A service can be a user journey, internal or external API, or some other boundary — essentially anything you care about setting a service level objective for. For example, in a service desk application one service might be creating a new ticket. That service may rely on a user service, a queue, a notification service, and a database service, all of which could additionally be defined as additional services in Nobl9.

A service may be composed of other services. When adding a service you can use labels to add additional metadata such as team ownership or upstream/downstream dependencies. Services can either be manually added in via the user interface or YAML, or can be automatically discovered from a datasource based on rules.

A service can have one or more SLOs defined for it. Every SLO created in Nobl9 must be tied to a service.

2. Add data sources for the service



You can have multiple data sources for your service. Configure Nobl9 to connect to these (one or multiple) data sources to collect all service data in real-time.

3. Define Service Level Objectives (SLOs)

With a service label and its data sources configured, define the thresholds for **Service Level Indicators (SLIs)**. Together with a time window these comprise a unique SLO.


Add an SLO

Perform these following steps to create an SLO in the web UI:

1. Navigate to  **Service Level Objectives** icon and click the  icon to start the **SLO Wizard**.
2. Follow the five-step configuration in the SLO wizard create an SLO.

3. Select a **Service** from the drop-down list to tag the service this SLO applies to.
4. Go to **Select Data Source and Metric** step and click the **Data Source** drop-down list to choose a data source.
5. Select a type of **Metric**. and enter a **Query**.
 - A **Threshold Metric** is a single time series evaluated against a threshold.
 - A **Ratio Metric** allows you to enter two time series to compare (for example, a count of good requests and total requests).
6. Enter a **Query, Good Query, or Total Query** for the metric you selected.
 - Example of a Threshold Metric Query:
 - Example of Ration Metric Query:

Description	Result
Web service or API	HTTPS responses with 2xx and 3xx status codes.
In a queue consumer	Sucessful processing of a message .
In aServerless and function-based architectures	successful completion of an invocation.
In batch	normal exit (for example, rc == 0) of the driving process or script.
In a browser application	completion of a user action without JavaScript errors.


7. Choose a **Rolling** or **Calendar-Aligned** time window in the **Define Time Window** section.
 - Rolling time windows are better for tracking *recent* user experience of a service.
 - Calendar-aligned windows are best suited for SLOs that are intended to map to business metrics that are measured on calendar-aligned basis, such as every calendar month, or every quarter.
8. **Define Error Budget Calculation and Objectives.** Click the drop-down list in **Error Budget Calculation Method** and select either **Occurences** or **Time Slices**. For more information, see the use case examples located in the last section of the Getting Started Guide.
9. Name your objective in the **Add Name, Alert Policy & Tags** section.
10. Click the drop-down list next to **Alert Policies** to sent an alert. If no alerts were created, navigate to the **Alert Policies** page and click the  icon to start the **Alert Policy Wizard**.
11. Enter a **Description**. Document relevant details or metadata for the SLI and SLO as description. As a best practice, it is recommended to add the team or owner details, or the purpose of creating this specific SLO. This may serve to provide a quick context about this SLO to any team member.

Use Cases of SLO Configurations

The following examples explain how to create SLOs for sample services using `sloctl`.

A Typical Example of a Latency SLO for a RESTful Service

First, we want to pick an appropriate service level indicator to measure the latency of response from a RESTful service. In this example, let's assume our service runs in NGINX web server, and we're going to use a threshold-based approach to define acceptable behavior. For example, we want the service to respond in a certain amount of time.

 **Note:** *There are many ways to measure application performance. In this case we're giving an example of server-side measurement at the application layer (NGINX) but it might be advantageous for your application to measure this differently. For example, you might choose to measure performance at the client, or at the load balancer, or somewhere else. Your choice depends on what you are trying to measure or improve, as well as what data is currently available as usable metrics for the SLI.*

The threshold approach uses a single query, and we set thresholds or breaking points on the results from that query to define the boundaries of acceptable behavior. In the SLO YAML, we specify the indicator like this:

```
indicator:
  metricSource:
    name: my-prometheus-instance
    project: default
  rawMetric:
    prometheus:
      promql: server_requestMsec{job="nginx"}
```

In this example use Prometheus. The concepts are similar for other metrics stores. We recommend running the query against your Prometheus instance and reviewing the result data, so you can verify that the query returns what you expect, and so that you understand the units: whether it's returning latencies as milliseconds or fractions of a second, for example. This query seems to return data between 60 and 150 milliseconds with some occasional outliers.

Choosing a Time Window.

We need to choose whether we want a rolling or calendar-aligned window.

- Calendar-aligned windows are best suited for SLOs that are intended to map to business metrics that are measured on calendar-aligned basis, such as every calendar month, or every quarter.
- Rolling windows are better for tracking "recent" user experience of a service.

For our RESTful service, we'll be using the Rolling window SLO primarily to measure recent user experience and to make decisions about the risk of changes, releases, and how best to invest our engineering resources on a week-to-week or sprint-to-sprint basis. We want the "recent" period that

we're measuring to trail back long enough that our users would consider it recent behavior. We choose to go with a 28-day window, which has the advantage of containing an equal number of weekend days and weekdays as it rolls.

```
timeWindows:  
  - count: 28  
    isRolling: true  
    period:  
      begin: "2020-12-01T00:00:00Z"  
      unit: Day
```

Choosing a Budgeting Method

There are two budgeting methods to choose from: **Time Slices** and **Occurrences**.

Time Slices

In the Time Slices method, what we count (objective we measure) is how many good minutes were achieved (minutes where our system is operating within defined boundaries), compared to the total minutes in the window.

This is useful for some scenarios, but it has a disadvantage when we're looking at "recent user experience" as we are with this SLO. The disadvantage is that a bad minute that occurs during a low-traffic period (say, in the middle of the night for most of our users, when they are unlikely to even notice a performance issue) would penalize the SLO the same amount as a bad minute during peak traffic times.

Occurrences

The Occurrences method is well suited to this situation. Occurrences count good attempts (in this example, requests that are within defined boundaries) against the count of all attempts (this means all requests, including requests that perform outside of defined boundaries). Since total attempts are fewer during low-traffic periods, it automatically adjusts to lower traffic volume.

```
budgetingMethod: Occurrences
```

Establishing Thresholds

In this example we've talked to our product and support teams and can establish the following thresholds:

- The service has to respond fast enough that users don't see any lag in the web applications that use this service
- Our Product Manager thinks that 100ms (1/10th of a second) is a reasonable threshold for what qualifies at Okay latency. We want to try hit that 95% of the time, so we code the first threshold like this: `

- `budgetTarget: 0.95` `displayName: Laggy` `value: 100` ` This threshold requires that 95% of request complete within 100ms.

You can name each threshold however you want. We recommend naming them how a user of the service (or how another service that uses this service) might describe the experience at a given threshold. Typically, we use names that are descriptive adjectives of the experience when the threshold is not met. When the threshold is violated, we can say that the user's experience is "Laggy".

- Some requests fall outside of that 100ms range. We want to make an allowance for that, but we also want to set other thresholds so that we know that even in its worst moments, our service is performing acceptably, and/or that its worst moments are brief.

Let's define another threshold. In the above threshold, we allow 5% of requests to run longer than 100ms. We want most of that 5%, say 80% of the remaining 5% of the queries to still return within 1/4th of a second (250ms). That means 99% of the queries return within 250ms (95% +4%). Add a threshold like this:

```
- budgetTarget: 0.99
  displayName: Slow
  value: 250
```

This threshold requires that 99% of requests complete within 250ms.

- While that covers the bulk of requests, even within the 1% of requests that we allow to exceed 250ms, the vast majority of them should complete within half a second (500ms).. Even within the 1% of requests that we allow to exceed 250ms, we want to make sure the vast majority of them complete within half a second (500ms). Add a threshold like this: `
 - `budgetTarget: 0.999` `displayName: Painful` `value: 500` ` This threshold requires that 99.9% of requests complete within 500ms. Putting it all together, our SLO definition for the use cases looks like this:

```
- apiVersion: n9/v1alpha
  kind: SLO
  metadata:
    displayName: adminpageload
    name: adminpageload
    project: external
  spec:
    alertPolicies: []
    budgetingMethod: Occurrences
    description: ""
    indicator:
      metricSource:
```



```
    name: cooperlab
    projects: default
rawMetric:
  newRelic:
    nrql: SELECT average(duration) FROM SyntheticRequest WHERE monitorId =

timeSeries:
  objectives:
    - displayName: ok
      op: lt
      tag: external.adminpageload.70d000000
      target: 0.98
      value: 70

    - displayName: laggy
      op: lt
      tag: external.adminpageload.85d000000
      target: 0.99
      value: 85

    - displayName: poor
      op: lt
      tag: external.adminpageload.125d000000
      value: 125
      service: venderportal

timeWindows:
  - count: 1
    isRolling: true
    period:
      begin: "2021-03-08T06:46:08Z"
      end: "2021-03-08T07:46:08Z"
    unit: Hour
```